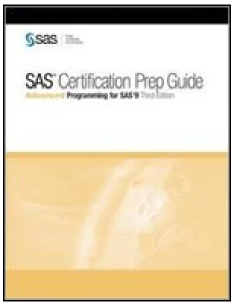


sas

SAS Certification Prep Guide

Advanced Programming for SAS 9 Third Edition

SAS Certification Prep Guide: Advanced Programming for SAS 9, Third Edition

by SAS Institute

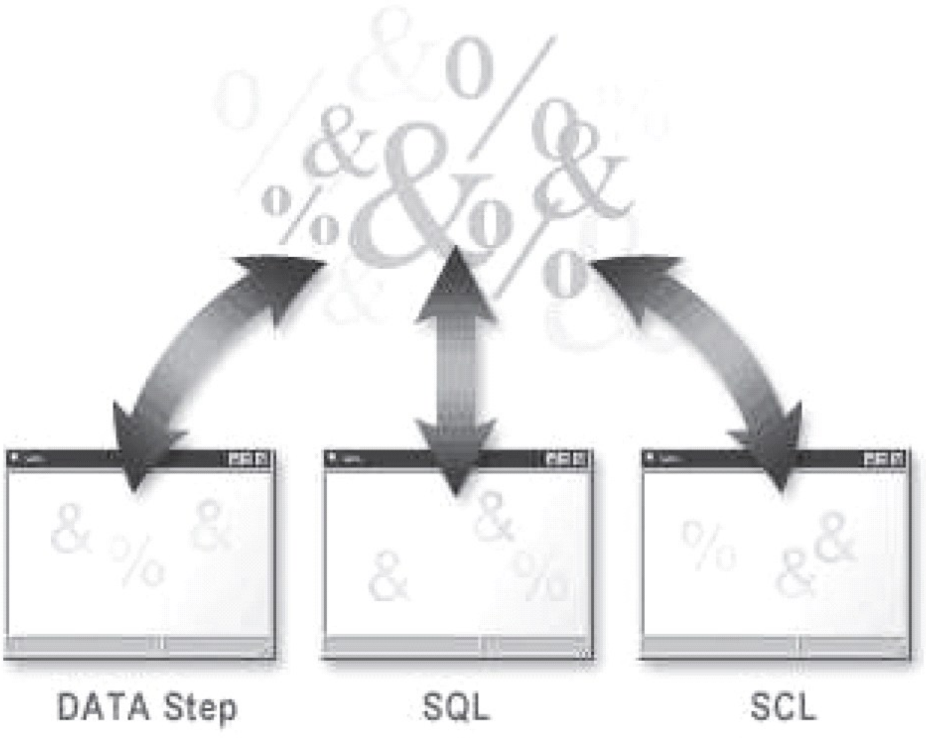
Chapter 10: Processing Macro Variables at Execution Time

Overview

Introduction

Because the macro facility performs its tasks before SAS programs execute, the information that the macro facility supplies does not depend on values that are accessed or computed during the execution of a SAS program. However, sometimes it is necessary to access or create macro variables during the execution of a SAS program. There are several methods that enable the macro facility to create or access macro variables *at execution time*. In this chapter, you learn to use macro variables during execution of

- a DATA step
- a PROC SQL step
- an SCL program.



Objectives

In this chapter, you learn to

- create macro variables during DATA step execution
- describe the difference between the SYMPUT routine and the %LET statement
- reference macro variables indirectly, using multiple ampersands for delayed resolution
- obtain the value of a macro variable during DATA step execution
- describe the difference between the SYMGET function and macro variable references
- create macro variables during PROC SQL execution
- store several values in one macro variable, using the SQL procedure

- create, update, and obtain the values of macro variables during the execution of an SCL program.

Prerequisites

Before beginning this chapter, you should complete the following chapters:

- "Performing Queries Using PROC SQL" on page 4
- "Creating and Managing Views Using PROC SQL" on page 260.
- "Introducing Macro Variables" on page 304

Creating a Macro Variable During DATA Step Execution

Overview

In many applications, you need to create macro variables *during DATA step execution*. You might need to create macro variables and to assign values to them based on

- data values in SAS data sets or in external files
- programming logic
- computed values.

For example, suppose you want to create a report that lists students who are enrolled in a specific course, according to data in the *Sasuser.All* data set. Suppose you want to include a footnote in your report to indicate whether any student fees are unpaid.

The following program uses SAS programming logic to determine which value is assigned to the macro variable **foot.** Then **foot** is referenced in the FOOTNOTE statement later in the program.

```
options symbolgen pagesize=30;
%let crsnum=3;
data revenue;
   set sasuser.all end=final;
   where course_number=&crsnum;
   total+1;
   if paid='Y' then paidup+1;
   if final then do;
      put total= paidup=; /* Write information
                               to the log. */
      if paidup<total then do;
         %let foot=Some Fees Are Unpaid;
      end;
      else do;
         %let foot=All Students Have Paid;
      end;
   end;
run;

proc print data=revenue;
   var student_name student_company paid;
   title "Payment Status for Course &crsnum";
   footnote "&foot";
run;
```

Running the program produces the following report:

payment status for course 3			
Obs	**student_Name**	**Student_Company**	**Paid**
1	Bills. Ms. Paulette	Reston Railway	Y
2	Chevarley, MS. Arlene	Motor Communicators	N

3	Clough. Ms. Patti	Reston Railway	N
4	Crace, Mr. Ron	Von Crump Seafood	Y
5	Davis. Mr. Bruce	Semi:Conductor	Y
6	Elsins. Ms. Marisa F.	SSS Inc.	N
7	Gandy. Dr. David	Paralegal Assoc.	Y
8	Gash, Ms. Hedy	QA Information Systems Center	Y
9	Haubold, Ms. Ann	Reston Railway	Y
10	Hudock, Ms. Cathy	So. Cal. Medical Center	Y
11	Kimble. Mr. John	Alforone Chemical	N
12	Kochen. Mr. Dennis	Reston Railway	Y
13	Larocque, Mr. Bret	Physicians IPA	Y
14	Licht. Mr. Bryan	SII	Y
15	McKnight, Ms. Maureen E.	Federated Bank	Y
16	Scannell, Ms. Robin	Amberly Corp.	N
17	Seitz, Mr Adam	Lomax Services	Y
18	Smith. Ms, Jan	Reston Railway	N
19	SUlzbach, Mr. Bill	Sailbest Ships	Y
20	Williams, Mr. Gene	Snowing Petroleum	Y
All Students Have Paid			

Although you can see that several students still have unpaid fees, the footnote indicates that all students have paid. Obviously, the footnote is wrong. That is, the macro variable **foot** resolves to the value *All Students Have Paid* when it should not do so. Look at the following example.

Example

In order to understand the problem with this example, you should consider how macro variable processing works in conjunction with SAS processing. Remember that when both macro language statements and SAS language statements occur in the same step, the *macro processor* executes macro language statements *before* any SAS language statements are executed.

Remember, you want to create a report that lists students who are enrolled in a specific course, according to data in the *Sasuser.All* data set, and you want to include a footnote in your report to indicate whether any student fees are unpaid. The following program uses SAS programming logic to determine which value is assigned to the macro variable **foot.** Then **foot** is referenced in the FOOTNOTE statement later in the program.

```
options symbolgen pagesize=30;
%let crsnum=3;
data revenue;
   set sasuser.all end=final;
   where course_number=&crsnum;
   total+1;
   if paid='Y' then paidup+1;
   if final then do;
      put total= paidup=; /* Write information
                           to the log. */
      if paidup<total then do;
         %let foot=Some Fees Are Unpaid;
      end;
      else do;
         %let foot=All Students Have Paid;
      end;
    end;
run;

proc print data=revenue;
   var student_name student_company paid;
   title "Payment Status for Course &crsnum";
   footnote "&foot";
run;
```

In this example, the first %LET statement inside the DATA step is passed to the macro processor as soon as the word scanner encounters it. The macro processor then creates a macro variable named **`foot`** in the symbol table and assigns the value *Some Fees Are Unpaid* to the variable.

The word scanner then continues to read the program and passes the second %LET statement in the DATA step to the macro processor as well. This time, the macro processor reassigns the value *All Students Have Paid* to **`foot`** in the symbol table.

When the RUN statement in the DATA step is encountered, SAS recognizes that the step is complete, and executes it. Remember that at this point the DATA step no longer includes any of the %LET statements (which have already been executed by the macro processor). Because the %LET statements are *always* processed by the macro processor *before* the DATA step is executed, the value of **`foot`** will always be whatever the last %LET statement assigns.

Here is a representation of the program that is processed by the DATA step compiler as a result of the above code.

Table 10.1: Code After Substitution

```
data revenue;
   set sasuser.all end=final;
   where course_number=3;
   total+1;
   if paid='Y' then paidup+1;
   if final then do;
      put total= paidup=;
      if paidup<total then do;
      end;
      else do;
      end;
   end;
run;
proc print data=revenue;
   var student_name student_company paid;
   title "Payment Status for Course 3";
   footnote "All Students Have Paid";
run;
```

We can solve this problem with the following information.

The SYMPUT Routine

The DATA step provides functions and a CALL routine that enable you to transfer information between an executing DATA step and the macro processor. You can use the SYMPUT routine to create a macro variable and to assign to that variable any value that is available in the DATA step.

General form, SYMPUT routine:

CALL SYMPUT*(macro-variable, text);*

where

macro-variable

is assigned the character value of *text*.

macro-variable and *text*

can each be specified as

- a literal, enclosed in quotation marks

- a DATA step variable
- a DATA step expression.

Note If *macro-variable* already exists, the value of *text* replaces the former value.

When you use the SYMPUT routine to create a macro variable in a DATA step, the macro variable is not actually created and assigned a value until the DATA step is *executed*. Therefore, you cannot successfully reference a macro variable that is created with the SYMPUT routine by preceding its name with an ampersand until after the step boundary that causes DATA step execution.

In the next few sections you will see several examples of how the SYMPUT routine can be used in different situations.

Using SYMPUT with a Literal

In the SYMPUT routine, you use a literal string for

- the first argument to specify an *exact name* for the name of the macro variable
- the second argument to specify the *exact character value* to assign to the macro variable.

To use a literal with the SYMPUT routine, you enclose the literal string in quotation marks.

CALL SYMPUT *('macro-variable', 'text');*

Example

Remember the previous example, in which you wanted to conditionally assign a value to the macro variable **`foot`** based on values that are generated during DATA step execution. You can use the SYMPUT routine with literal strings as both arguments in order to accomplish this.

```
options symbolgen pagesize=30;
%let crsnum=3;
data revenue;
   set sasuser.all end=final;
   where course_number=&crsnum;
   total+1;
   if paid='Y' then paidup+1;
   if final then do;
   if paidup<total then do;
      call symput('foot','Some Fees Are Unpaid');
   end;
   else do;
      call symput('foot','All Students Have Paid');
   end;
end;
run;

proc print data=revenue;
   var student_name student_company paid;
   title "Payment Status for Course &crsnum";
   footnote "&foot";
run;
```

This time, the value assigned to **`foot`** is either *Some Fees Are Unpaid* or *All Students Have Paid*, depending on the value of the DATA step variable **`Paidup`**, because the value is assigned during the execution of the DATA step. When you submit this code, you get the following output.

Payment Status for Course 3

Obs	Student_Name	Student_Company	Paid
1	Bills, Ms. Paulette	Reston Railway	Y
2	Chevarley, Ms. Arlene	Motor Communications	N

3	Clough, Me. Patti	Reston Railway	N
4	Crace, Mr. Ron	Von Crump Seafood	Y
5	Davis, Mr. Bruce	Semi: Conductor	Y
6	Elsins, Ms. Marisa F.	SSS Inc.	N
7	Gandy. Dr. David	Paralegal Assoc.	Y
8	Gash, Ms Hedy	QA Information Systems Center	Y
9	Haubold, Ms. Ann	Reston Railway	Y
10	Hudock, Ms. Cathy	So. Cal. Medical Center	Y
11	Kimble, Mr. John	Alforone Chemical	N
12	Kochen. Mr. Dennis	Reston Railway	Y
13	Larocque, Mr. Bret	Physicians IPA	Y
14	Licht, Mr. Bryan	SII	Y
15	McKnight, Ms. Maureen E.	Federated Bank	Y
16	Scannell, Ms. Robin	Amberly Corp.	N
17	Seitz, Mr. Adam	Lomax Services	Y
18	Smith. Ms. Jan	Reston Railway	N
19	Sulzbach. Mr. Bill	Sailbest Ships	Y
20	Williams. Mr. Gene	Snowing Petroleum	Y
Some Fees Are Unpaid			

Using SYMPUT with a DATA Step Variable

You can assign the value of a DATA step variable as the value for a macro variable by using the DATA step variable's name as the second argument to the SYMPUT routine.

To use a DATA step variable as the value for a macro variable in the SYMPUT routine, you place the name of the DATA step variable after the name of the macro variable, separated by a comma. You do *not* enclose the name of the DATA step variable in quotation marks.

CALL SYMPVT(*'macro-variable', DATA-step-variable);*

This form of the SYMPUT routine creates the macro variable named *macro-variable* and assigns to it the current value of *DATA-step-variable*.

When you use a DATA step variable as the second argument,

- a maximum of 32,767 characters can be assigned to the receiving macro variable.
- any leading or trailing blanks that are part of the DATA step variable's value are stored in the macro variable.
- values of numeric variables are *automatically converted* to character values, using the BEST12. format.

Caution If you enclose the DATA step variable name in quotation marks, SAS interprets the name as a *literal value* rather than as a variable name, and the DATA step variable's value is *not resolved*.

Example

Once again, suppose you want to create a report about students who are enrolled in a particular course. This time, suppose you want to add a title that contains the course title and the course number, and you want to include a footnote that summarizes how many students have paid their fees.

In this example, a DATA step variable named **`paidup`** records the number of students that have paid, and a DATA step variable named **`total`** records the total number of students who are registered for the class. Macro variables are created to record the values of **`paidup`**, the value of **`total`**, and the value of **`Course_title.`** These macro variables are referenced later in the program.

```
%let crsnum=3;
data revenue;
   set sasuser.all end=final;
   where course_number=&crsnum;
   total+1;
   if paid='Y' then paidup+1;
   if final then do;
      call symput('numpaid',paidup);
      call symput('numstu',total);
      call symput('crsname',course_title);
   end;
run;
proc print data=revenue noobs;
   var student_name student_company paid;
   title "Fee Status for &crsname (#&crsnum)";
   footnote "Note: &numpaid Paid out of &numstu Students";
run;
```

This time the footnote shows the correct information for how many students have paid.

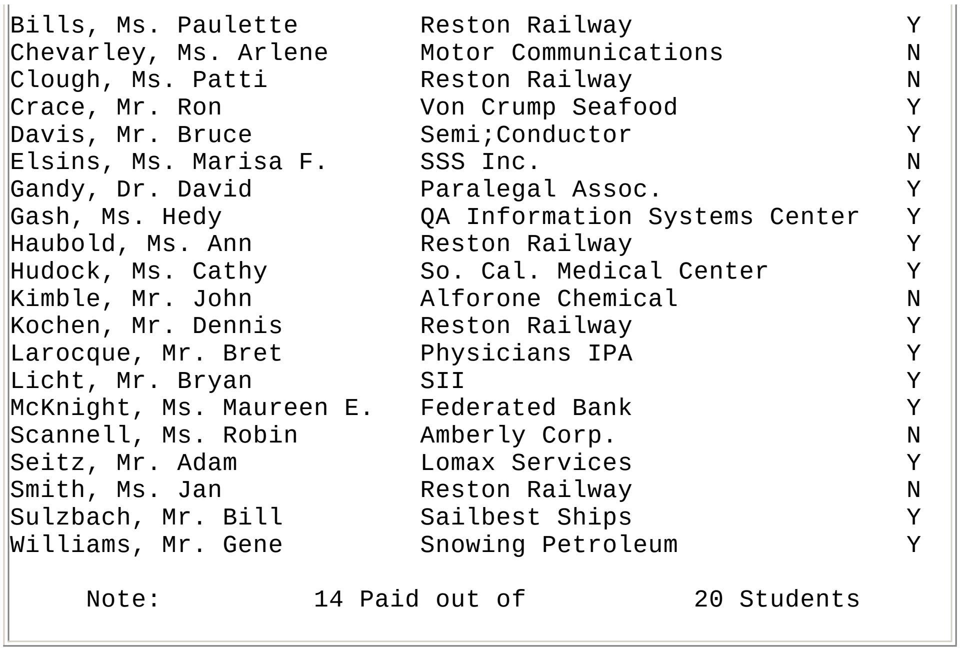
Fee Status for Local Area Networks (#3)		
Student_Name	**Student_Company**	**Paid**
Bills, Ms. Paulette	Reston Railway	Y
Chevarley, Ms. Arlene	Motor Communications	N
Clough, Ms. Patti	Reston Railway	N
Crace, Mr. Ron	Von Crump Seafood	Y
Davis, Mr. Bruce	Semi;Conductor	Y
Elsins, Ms. Marisa F	SSS Inc.	N
Gandy. Dr David	Paralegal Assoc.	Y
Gash, Ms. Hedy	QA Information Systems Center	Y
Haubold Ms. Ann	Reston Railway	Y
Hudock, Ms. Cathy	So. Cal. Medical Center	Y
Kimble, Mr. John	Alforone Chemical	N
Kochen, Mr. Dennis	Reston Railway	Y
Larocque, Mr. Bret	Physicians IPA	Y
Licht, Mr. Bryan	SII	Y
McKnight Ms Maureen E.	Federated Bank	Y
Scannell, Ms. Robin	Amberly Corp.	N
Seitz, Mr. Adam	Lomax Services	Y
Smith, Ms. Jan	Reston Railway	N
Sulzbach, Mr. Bill	Sailbest Ships	Y
Williams, Mr. Gene	Snowing Petroleum	Y
Note: 14 Paid out. of 20 Students		

Using CALL SYMPUT with DATA Step Expressions

If you had run the last example using *listing output* rather than HTML output, you would have seen extra blanks in the title between the course title and the course number, as well as in the footnote.

Table 10.2: SAS Listing Output

```
Fee Status for Local AreaNetworks (#3)


Student_Name               Student_Company                Paid
```



```
Bills, Ms. Paulette        Reston Railway                Y
Chevarley, Ms. Arlene      Motor Communications          N
Clough, Ms. Patti          Reston Railway                N
Crace, Mr. Ron             Von Crump Seafood             Y
Davis, Mr. Bruce           Semi;Conductor                Y
Elsins, Ms. Marisa F.      SSS Inc.                      N
Gandy, Dr. David           Paralegal Assoc.              Y
Gash, Ms. Hedy             QA Information Systems Center Y
Haubold, Ms. Ann           Reston Railway                Y
Hudock, Ms. Cathy          So. Cal. Medical Center       Y
Kimble, Mr. John           Alforone Chemical             N
Kochen, Mr. Dennis         Reston Railway                Y
Larocque, Mr. Bret         Physicians IPA                Y
Licht, Mr. Bryan           SII                           Y
McKnight, Ms. Maureen E.   Federated Bank                Y
Scannell, Ms. Robin        Amberly Corp.                 N
Seitz, Mr. Adam            Lomax Services                Y
Smith, Ms. Jan             Reston Railway                N
Sulzbach, Mr. Bill         Sailbest Ships                Y
Williams, Mr. Gene         Snowing Petroleum             Y

     Note:          14 Paid out of           20 Students
```

You do not see these blanks If you are using HTML output, but they are still stored in the value of your macro variable.

Remember that when a DATA step variable is used as the second argument in a SYMPUT routine, any leading or trailing blanks that are part of the DATA step variable's value are stored in the macro variable. Because the value of a macro variable is *always* a text string, numeric variables are automatically converted using the BEST12. format, and blanks are stored as part of the macro variable's value. In order to avoid including extra blanks, you need to use a DATA step function to remove them.

In these situations you can use DATA step functions before the SYMPUT routine executes, in order to

- left-align character strings that have been created by numeric-to-character conversions
- remove extraneous leading and trailing blanks.

Often you will want to combine several DATA step functions in order to create a DATA step expression as the second argument of the SYMPUT routine.

CALL SYMPUT(*'macro-variable', expression);*

> **Note** A DATA step expression can be any combination of DATA step functions, DATA step variables, constants, and logical or arithmetic operators that resolves to a character or numeric constant.

When you use a DATA step expression as the second argument, its current value is evaluated according to the following rules:

- Numeric expressions are automatically converted to character constants using the BEST12. format.
- The resulting value can be up to 32,767 characters long.
- Any leading or trailing blanks that are part of the expression are stored in the macro variable.

Example

In order to remove the extra blanks from the title and footnote of the previous example, you can use DATA step functions. To remove trailing blanks from **`crsname`**, you can use the TRIM function. To remove leading and trailing blanks from the macro variables **`numstu`** and **`numpaid`**, you can use the STRIP function.

```
%let crsnum=3;
data revenue;
   set sasuser.all end=final;
   where course_number=&crsnum;
   total+1;
   if paid='Y' then paidup+1;
   if final then do;

      call symput('numpaid',strip(paidup));
      call symput('numstu',strip(total));
      call symput('crsname',trim(course_title));
   end;
run;

proc print data=revenue noobs;
   var student_name student_company paid;
   title "Fee Status for &crsname (#&crsnum)";
   footnote "Note: &numpaid Paid out of &numstu Students";
run;
```

Table 10.3: SAS Listing Output

```
                 Fee Status for Local Area Networks (#3)


NAME                            COMPANY                      PAID

Bills, Ms. Paulette             Reston Railway                Y
Chevarley, Ms. Arlene           Motor Communications          N
Clough, Ms. Patti               Reston Railway                N
Crace, Mr. Ron                  Von Crump Seafood             Y
Davis, Mr. Bruce                Semi;Conductor                Y
Elsins, Ms. Marisa F.           SSS Inc.                      N
Gandy, Dr. David                Paralegal Assoc.              Y
Gash, Ms. Hedy                  QA Information Systems Center Y
Haubold, Ms. Ann                Reston Railway                Y
Hudock, Ms. Cathy               So. Cal. Medical Center       Y
Kimble, Mr. John                Alforone Chemical             N
Kochen, Mr. Dennis              Reston Railway                Y
Larocque, Mr. Bret              Physicians IPA                Y
Licht, Mr. Bryan                SII                           Y
McKnight, Ms. Maureen E.        Federated Bank                Y
Scannell, Ms. Robin             Amberly Corp.                 N
Seitz, Mr. Adam                 Lomax Services                Y
Smith, Ms. Jan                  Reston Railway                N
Sulzbach, Mr. Bill              Sailbest Ships                Y
Williams, Mr. Gene              Snowing Petroleum             Y

                 Note: 14 Paid out of 20 Students
```

PUT Function

Remember that the values of macro variables are *always* character strings. You have seen that in the DATA step the SYMPUT routine will perform automatic numeric-to-character conversion on any numeric value that you attempt to assign to a macro variable. Messages are written to the SAS log to alert you that automatic conversion has occurred. Remember that the SYMPUT routine automatically uses the BEST12. format for the conversion.

Sometimes you might want to have explicit control over the numeric-to-character conversion. The PUT function returns a character string that is formed by writing a value with a specified format.

You can use the PUT function to

- perform explicit numeric-to-character conversions
- format the result of a numeric expression.

General form, PUT function:

PUT*(sourceformat.)*

where

source

is a constant, a variable, or an expression (numeric or character).

format.

is any SAS format or user-defined format, which determines

- the length of the resulting string
- whether the string is right-or left-aligned.

source and format.

must be the same type (numeric or character).

Example

Suppose you want to create a report that shows the amount of fees that are unpaid for a specific course. In the following example, you use the SYMPUT routine to format the value of the numeric variable **Begin_date** with the MMDDYY10. format and assign that value to the macro variable **date.** Then you also use another call to the SYMPUT routine to format the result of an expression involving **Fee, total**, and **paidup** as a dollar amount and assign that value to the macro variable **due.**

```
%let crsnum=3;
data revenue;
   set sasuser.all end=final;
   where course_number=&crsnum;
   total+1;
   if paid='Y' then paidup+1;
   if final then do;
     call symput('crsname',trim(course_title));
     call symput('date',put(begin_date,mmddyy10.));
     call symput('due',strip(put(fee*(total-paidup),dollar8.)));
   end;
run;
```

You can use the macro variables **date** and **due** in a PROC PRINT step to create your report. The values of these macro variables appear in the report with the formatting that you assigned to them when you created them.

```
proc print data=revenue;
   var student_name student_company paid;
   title "Fee Status for &crsname (#&crsnum) Held &date";
   footnote "Note: &due in Unpaid Fees";
run;
```

Fee Status for Local Area Networks (#3) Held 01/08/2001

Obs	Student_Name	Student_Company	Paid
1	Bills, Ms. Paulette	Reston Railway	Y
2	Chevarley, Ms. Arlene	Motor Communications	N
3	Clough, Ms. Patti	Reston Railway	N
4	Crace, Mr. Ron	Von Crump Seafood	Y
5	Davis, Mr. Bruce	Semi;Conductor	Y
6	Elsins, Ms. Marisa F	SSS Inc.	N
7	Gandy, Dr. David	Paralegal Assoc.	Y
8	Gash, Ms. Hedy	QA Information Systems Center	Y
9	Haubold, Ms. Ann	Reston Railway	Y
10	Hudock, Ms. Cathy	So. Cal Medical Center	Y
11	Kimble, Mr. John	Alforone Chemical	N

12	Kochen, Mr. Dennis	Reston Railway	Y
13	Larocque, Mr. Bret	Physicians IPA	Y
14	Licht, Mr. Bryan	SII	Y
15	McKnight, Ms Maureen E.	Federated Bank	Y
16	Scannell, Ms. Robin	Amberly Corp.	N
17	Seitz, Mr. Adam	Lomax Services	Y
18	Smith, Ms. Jan	Reston Railway	N
19	Sulzbach, Mr. Bill	Sailbest Ships	Y
20	Williams, Mr. Gene	Snowing Petroleum	Y
Note: $3,900 in Unpaid Fees			

The SYMPUTX Routine

The SYMPUTX routine is very similar to the SYMPUT routine. In addition to creating a macro variable and assigning a value to it, the SYMPUTX routine also automatically removes leading and trailing blanks from both arguments.

General form, SYMPUTX routine:

CALL SYMPVTX(*macro-variable,expression);*

where

macro-variable

is assigned the character value of *expression*, and any leading or trailing blanks are removed from both *macro-variable* and *expression*.

macro-variable and *expression*

can each be specified as

- a literal, enclosed in quotation marks
- a DATA step variable
- a DATA step expression.

Note If *macro-variable* already exists, the value of *expression* replaces the former value.

Example

Remember the example where you created a report about students who are enrolled in a particular course. This time, suppose you want the title to contain the course name and the course number, as well as the date on which the course was held. Also, you want the footnote to list the current amount of unpaid fees for the course.

In this example, three macro variables are created. The macro variable **`csrname`** records the value of the DATA step variable **`Course_title.`** The macro variable **`date`** records the value of the DATA step variable **`Begin_date`** in MMDDYY10. format. Finally, the macro variable **`due`** uses the values of the DATA step variables **`paidup, total`**, and **`fee`** to record the current amount of unpaid fees in DOLLAR8. format. These macro variables are referenced later in the program in the title and footnote statements.

```
%let crsnum=3;
data revenue;
   set sasuser.all end=final;
   where course_number=&crsnum;
   total+1;
   if paid='Y' then paidup+1;
   if final then do;
      call symputx('crsname',course_title);
      call symputx('date',put(begin_date,mmddyy10.));
      call symputx('due',put(fee*(total-paidup),dollar8.));
    end;
run;
proc print data=revenue;
   var student_name student_company paid;
   title "Fee Status for &crsname (#&crsnum) Held &date";
   footnote "Note: &due in Unpaid Fees";
run;
```

Fee Status for Local Area Networks (#3) Held 01/08/2001

Obs	Student_Name	Student_Company	Paid
1	Bills Ms. Paulette	Reston Railway	Y
2	Chevarley, Ms. Arlene	Motor Communications	N
3	Clough, Ms. Patti	Reston Railway	N
4	Crace, Mr. Ron	Von Crump Seafood	Y
5	Davis, Mr. Bruce	Semi;conductor	Y
6	Elsins, Ms. Marisa F.	SSS Inc.	N
7	Gandy, Dr. David	Paralegal Assoc	Y
8	Gash, Ms. Hedy	QA Information Systems Center	Y
9	Haubold, Ms. Ann	Reston Railway	Y
10	Hudock, Ms. Cathy	So. Cal. Medical Center	Y
11	Kimble, Mr. John	Alforone Chemical	N
12	Kochen, Mr. Dennis	Reston Railway	Y
13	Larocque, Mr. Bret	Physicians IPA	Y
14	Licht, Mr, Bryan	SII	Y
15	McKnight, Ms. Maureen E.	Federated Bank	Y
16	Scannell, Ms. Robin	Amberly Corp.	N
17	seitz, Mr, Adam	Lomax Services	Y
18	Smith. Ms, Jan	Reston Railway	N
19	Sulzbach, Mr. Bill	Sailbest Ships	Y
20	Williams, Mr. Gene	Snowing Petroleum	Y

Note: $3.900 in Unpaid Fees

Creating Multiple Macro Variables During DATA Step Execution

Creating Multiple Macro Variables with CALL SYMPUT

Sometimes you might want to create *multiple* macro variables within *one* DATA step. For example, suppose you want to write a program that will list all of the scheduled dates for a particular course, using a macro variable to record the title of the course.

```
%let crsid=C005;
data _null_;
   set sasuser.courses;
   where course_code='&crsid";
   call symput('title',trim(course_title));
run;

proc print data=sasuser.schedule noobs label;
   where course_code='&crsid";
   var location begin_date teacher;
   title1 "Schedule for &title";
   options nodate nonumber;
run;
```

In this example, the value of the data set variable **`Course_title`** for the course whose **`Course_code`** is *C005* is assigned as a value for the macro variable **`title.`** The value **`_null_`** on the data statement is used because we do not need a data set to be created in this example.

In order to create a listing for a different course, you would need to change the %LET statement and resubmit the DATA step to assign a new value to **`title.`** Then you would need to resubmit the PROC PRINT step. Although you would need to resubmit both the DATA step and the PROC PRINT step, these two steps would be identical to the steps that you submitted for the first report. This is an extremely inefficient program.

```
%let crsid=C004;
data _null_;
   set sasuser.courses;
   where course_code='&crsid";
   call symput('title',trim(course_title));
run;

proc print data=sasuser.schedule noobs label;
   where course_code='&crsid";
   var location begin_date teacher;
   title1 "Schedule for &title";
   options nodate nonumber;
run;
```

Instead of executing separate DATA steps to update the same macro variable, you can create related macro variables in one DATA step. To create multiple macro variables, you use the SYMPUT routine with DATA step expressions for *both* arguments.

General form, SYMPUT routine with DATA step expressions:

CALL SYMPUT*(expressionl, expression2);*

where

expression1

evaluates to a character value that is a valid macro variable name. This value should change each time you want to create another macro variable.

expression2

is the value that you want to assign to a specific macro variable.

Example

In this example, you use one call to the SYMPUT routine in order to create one macro variable for *each value* of the DATA step variable **`Course_code`** and to assign the corresponding value of **`Course_title`** to each macro variable. That is, for each observation in *Sasuser.Courses*, the macro processor will create a new macro variable. The new macro variable will have the same name as the value of the data set variable

`Course_code` for that observation. The value of the new macro variable will be the value of the data set variable **`Course_title`** for that observation.

```
data _null_;
   set sasuser.courses;
   call symput(course_code,trim(course_title));
run;
%put _user_;
```

The SAS log shows that six observations were read from the data set *Sasuser.Courses* and that six global macro variables were created and were assigned values.

Table 10.4: SAS Log

```
2   data _null_;
3      set sasuser.courses;
```

```
4      call symput(course_code, trim(course_title));
5   run;

NOTE: There were 6 observations read from the dataset
      SASUSER.COURSES.
NOTE: DATA statement used:
      real time           0.52 seconds
      cpu time            0.13 seconds
7 %put _user_;
GLOBAL C006 Computer Aided Design
GLOBAL C001 Basic Telecommunications
GLOBAL C002 Structured Query Language
GLOBAL C003 Local Area Networks
GLOBAL C004 Database Design
GLOBAL C005 Artificial Intelligence
```

You can then use these new macro variables to print listings of information for various courses, using only one DATA step, as follows:

```
data _null_;
   set sasuser.courses;
   call symput(course_code,trim(course_title));
run;

%let crsid=C005;
proc print data=sasuser.schedule noobs label;
   where course_code="&crsid";
   var location begin_date teacher;
   title1 "Schedule for &c005";
run;

%let crsid=C002;
proc print data=sasuser.schedule noobs label;
   where course_code="&crsid";
   var location begin_date teacher;
   title1 "Schedule for &c002";
run;
```

This is the output from the first PROC PRINT step.

Schedule far Artificial Intelligence

Location	Begin	Instructor
Dallas	26FEB2001	Hallis, Dr. George
Boston	17SEP2001	Tally, Ms. Julia
Seattle	25FEB2002	Hallis, Dr. George

This is the output from the second PROC PRINT step.

Schedule for Structured Query Language

Location	Begin	Instructor
Dallas	04DEC2000	Wickam, Dr. Alice
Boston	11JUN2001	Wickam, Dr. Alice
Seattle	03DEC2001	Wickam, Dr. Alice

The program in this section is more efficient than the program shown in the previous section since the *Sasuser. Courses* data set is read only once in the latest example. However, there is still room for improvement.

Referencing Macro Variables Indirectly

Introduction

In the last example, you saw how to use the SYMPUT routine to create a series of macro variables whose names are based on the values of **Course_code.** However, you still needed to modify the TITLE statement in each PROC PRINT step in order to print output for each course.

Suppose you want to write a PROC PRINT step that you can reuse without any modification to print information about each course. You can do this by using an *indirect reference* in the TITLE statement.

```
data _null_;
   set sasuser.courses;
   call symput(course_code,trim(course_title));
run;

%let crsid=C002;
proc print data=sasuser.schedule noobs label;
   where course_code="&crsid";
   var location begin_date teacher;
   title1 "Schedule for ???";
run;
```

In the example above, the macro variable **C002** (as created by the SYMPUT routine) has a value of *Structured Query Language*. Therefore, the TITLE statement should reference a macro variable that will resolve to *Structured Query Language*. Remember that you want this reference to be flexible enough to apply to any of the macro variables that the SYMPUT routine creates, such as **C003** or **C004**, by changing only the %LET statement.

To obtain the value *Structured Query Language*, you need to indirectly reference the macro variable **C002** through a reference to the macro variable **crsid.** If the value of the macro variable **crsid** is *C002*, then you need to proceed in several steps:

1. Resolve the macro variable **crsid** to the value *C002*.
2. Attach an ampersand (**&**) to the front of the resolved value in order to create a new reference **(&C002).**
3. Resolve the resulting macro variable reference to the value *Structured Query Language*.

This sequence seems to imply that you should use the reference **&&crsid** to convert the value of the macro variable **crsid** to the corresponding course description. However, the Forward Re-Scan rule indicates that this is not the correct solution.

The Forward Re-Scan Rule

The Forward Re-Scan rule can be summarized as follows:

- When multiple ampersands or percent signs precede a name token, the macro processor resolves two ampersands (**&&**) to one ampersand (**&**), and re-scans the reference.
- To re-scan a reference, the macro processor scans and resolves tokens from left to right from the point where multiple ampersands or percent signs are coded, until no more triggers can be resolved.

According to the Forward Re-Scan rule, you need to use *three ampersands* in front of a macro variable name when its value matches the name of a second macro variable. This indirect reference resolves to the value of the second macro variable.

Example

Suppose you want to use the macro variable **crsid** to indirectly reference the macro variable **C002.**

Global Symbol Table	
C001	*Basic Telecommunications*
C002	*Structured Query Language*
C003	*Local Area Networks*
C004	*Database Design*

C005	*Artificial Intelligence*
C006	*Computer Aided Design*
CRSID	***C002***

The following table shows several references along with their resolved values.

Reference	Scan	Resolved Value	Re-scan	Resolved Value
&Crsid	→	C002	no re-scan	
&&Crsid	→	&Crsid	→	C002
&&&Crsid	→	&C002	→	Structured Query Language

By preceding a macro variable reference with two ampersands, you delay the resolution of the reference until the second scan. The first time the reference is scanned, only the double ampersands will be resolved (to one ampersand). In order to create an indirect reference (a reference whose value is a reference to a different macro variable), you must use three ampersands. Therefore, to use an indirect reference that resolves to *Structured Query Language*, the original reference must be **`&&&crsid`.**

Example

You can use indirect referencing to improve the last example. By using an indirect reference to the macro variable whose name is the same as the current value of the macro variable crsid, you can write a PROC PRINT step that you can reuse without modification in order to print a report for each different course.

```
options symbolgen;
data _null_;
   set sasuser.courses;
   call symput(course_code, trim(course_title));
run;

%let crsid=C005;
proc print data=sasuser.schedule noobs label;
   where course_code="&crsid";
   var location begin_date teacher;
   title1 "Schedule for &&&crsid";
run;

%let crsid=C002;
proc print data=sasuser.schedule noobs label;
   where course_code="&crsid";
   var location begin_date teacher;
   title1 "Schedule for &&&crsid";
run;
```

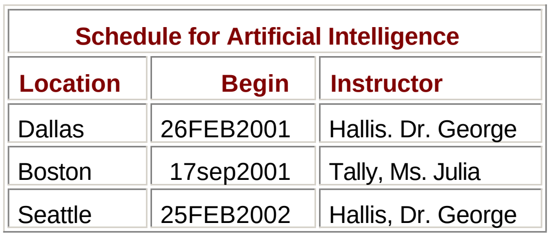
The SAS log shows the steps that lead to the resolution of these macro variables for each PROC PRINT step.

Table 10.5: SAS Log

```
43   options symbolgen;
44   data _null_;
45      set sasuser.courses;
46      call symput(course_code, trim(course_title));
47   run;
NOTE: There were 6 observations read from the dataset
      SASUSER.COURSES.
NOTE: DATA statement used:
      real time             0.07 seconds
      cpu time              0.05 seconds

48
49   %let crsid=C005;
50   proc print data=sasuser.schedule noobs label;
51      where course_code="&crsid";
SYMBOLGEN: Macro variable CRSID resolves to C005
52      var location begin_date teacher;
```

```
SYMBOLGEN: && resolves to &.
SYMBOLGEN: Macro variable CRSID resolves to C005
SYMBOLGEN: Macro variable C005 resolves to Artificial
           Intelligence
53      title1 "Schedule for &&&crsid";
54   run;

NOTE: There were 3 observations read from the dataset
      SASUSER.SCHEDULE.
      WHERE course_code='C005';
NOTE: PROCEDURE PRINT used:
      real time            0.09 seconds
      cpu time             0.04 seconds

55
56   %let crsid=C002;
57   proc print data=sasuser.schedule noobs label;
58      where course_code="&crsid";

SYMBOLGEN: Macro variable CRSID resolves to C002
59      var location begin_date teacher;
SYMBOLGEN: && resolves to &.
SYMBOLGEN: Macro variable CRSID resolves to C002
SYMBOLGEN: Macro variable C002 resolves to Structured
           Query Language
60      title1 "Schedule for &&&crsid";
61   run;

NOTE: There were 3 observations read from the dataset
      SASUSER.SCHEDULE.
      WHERE course_code='C002';
NOTE: PROCEDURE PRINT used:
      real time            0.06 seconds
      cpu time             0.04 seconds
```

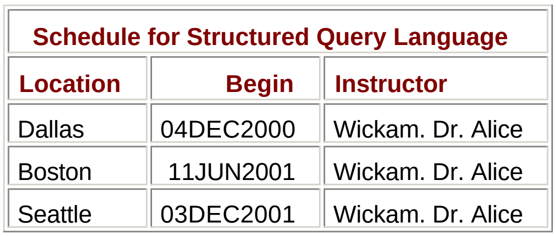
This is the output from the first PROC PRINT step.



Schedule for Artificial Intelligence

Location	Begin	Instructor
Dallas	26FEB2001	Hallis. Dr. George
Boston	17sep2001	Tally, Ms. Julia
Seattle	25FEB2002	Hallis, Dr. George

This is the output from the second PROC PRINT step.



Schedule for Structured Query Language

Location	Begin	Instructor
Dallas	04DEC2000	Wickam. Dr. Alice
Boston	11JUN2001	Wickam. Dr. Alice
Seattle	03DEC2001	Wickam. Dr. Alice

Note that the PROC PRINT steps that produced these reports were identical. Only the %LET statement that precedes each PROC PRINT step and the resolved values of the macro variables changed.

Indirect referencing is especially useful when you are working with a series of related macro variables. In "Introducing Macro Variables" on page 304, you learned how to combine multiple macro variable references in order to build new tokens. You can combine indirect macro variable references with other macro variable references as well. That is, you can use *two ampersands* in a reference when the *value* of one macro variable matches_part *of the name* of a second macro variable.

Example

You can create a series of macro variables, **`teach1`** to **`teach`*`n`***, each containing the name of the instructor who is assigned to a specific course.

```
options symbolgen;
data _null_;
   set sasuser.schedule;
   call symput('teach'||left(course_number),
               trim(teacher));
run;
```

> **Note** The concatenation operator **`||`** combines text. In the example above, the literal string **`teach`** is concatenated to the text that results from left-aligning the resolved value of the variable **`Course_number`.**

Global Symbol Table	
TEACHI	Hallis, Dr. George
TEACH2	Wickam, Dr. Alice
TEACH3	Forest, Mr. Peter

Then, you can reference one of these variables when a course number is designated. If you designate a course number in a %LET statement, you can use multiple ampersands in order to create a reference to the **`teach`*`n`*** macro variable that corresponds to the current course number.

```
%let crs=3;
proc print data=sasuser.register noobs;
   where course_number=&crs;
   var student_name paid;
   title1 "Roster for Course &crs";
   title2 "Taught by &&teach&crs";
run;
```

The SAS log shows the steps that lead to the resolution of the reference **`&&teach&crs`.**

Table 10.6: SAS LOG

```
65   %let crs=3;
66   proc print data=sasuser.register noobs;
67      where course_number=&crs;
SYMBOLGEN:  Macro variable CRS resolves to 3
68      var student_name paid;
SYMBOLGEN:   Macro variable CRS resolves to 3
69     title1 "Roster for Course &crs";
SYMBOLGEN: && resolves to &.
SYMBOLGEN: Macro variable CRS resolves to 3
SYMBOLGEN: Macro variable TEACH3 resolves to
           Forest, Mr. Peter
70     title2 "Taught by &&teach&crs";
71   run;
```

This is the output from the example.

Roster for Course 3

Taught by Forest, Mr. Peter

Student_Name	Paid
Bills, Ms. Paulette	Y
Chevarley, Ms. Ariene	N
Clough, Ms. Patti	N
Crace, Mr. Ron	Y
Davis, Mr. Bruce	Y

Elsins, Ms. Marisa F.	N
Gandy, Dr. David	Y
Gash, Ms. Hedy	Y
Haubold, Ms. Ann	Y
Hudock, Ms. Cathy	Y
Kimble, Mr. John	N
Kochen, Mr. Dennis	Y
Larocque, Mr. Bret	Y
Licht. Mr. Bryan	Y
McKnight, Ms. Maureen E.	Y
Scannell, Ms. Robin	N
Seitz, Mr. Adam	Y
Smith, Ms. Jan	N
SUizbcach, Mr. Bill	Y
Williams, Mr. Gene	Y

Obtaining Macro Variable Values During DATA Step Execution

The SYMGET Function

Earlier you learned how to use the SYMPUT routine to create macro variables in a DATA step. You are also familiar with using a macro variable reference such as **`&macvar`** to obtain the value of a macro variable *before* a DATA step executes. Now, suppose you want to obtain the value of a macro variable *during* DATA step execution. You can obtain a macro variable's value during DATA step execution by using the SYMGET function. The SYMGET function returns the value of an existing macro variable.

General form, SYMGET function:

SYMGET *(macro-variable)*

where

macro-variable

can be specified as one of the following:

- a macro variable name, enclosed in quotation marks
- a DATA step variable name whose value is the name of a macro variable
- a DATA step character expression whose value is the name of a macro variable.

Example

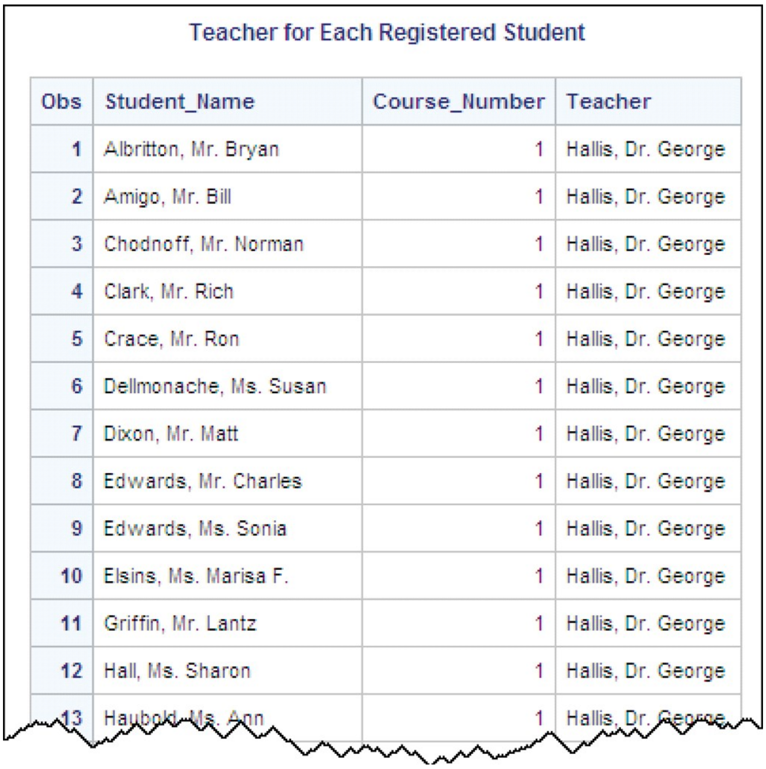
You can use the SYMGET function to obtain the value of a different macro variable for each iteration of aDATA step. In this example, the data set variable **`Teacher`** is assigned the value of the macro variable **`teach`*`n`*** for each observation in the *Sasuser.Register* data set, where *n* is the value of the data set variable **`Course_number`** for that observation.

Note This example assumes that a macro variable named **`teach`*`n`*** has already been created for each observation in *Sasuser.Register*.

```
data teachers;
   set sasuser.register;
   length Teacher $ 20;
   teacher=symget('teach'||left(course_number));
run;

proc print data=teachers;
   var student_name course_number teacher;
   title1 "Teacher for Each Registered Student";
run;
```

Global Symbol Table	
TEACHl	*Hallis, Dr. George*
TEACH2	*Wickam, Dr Alice*
TEACH3	*Foresf, Mr. Peter*
CRS	*3*

Part of the SAS output that this program creates is shown below. Notice that the new data set *Teachers* contains a variable named **`Teacher`** and that the values of this variable are the same as the values of the macro variables **`teach1-teach3`** in the global symbol table above.



Teacher for Each Registered Student

Obs	Student_Name	Course_Number	Teacher
1	Albritton, Mr. Bryan	1	Hallis, Dr. George
2	Amigo, Mr. Bill	1	Hallis, Dr. George
3	Chodnoff, Mr. Norman	1	Hallis, Dr. George
4	Clark, Mr. Rich	1	Hallis, Dr. George
5	Crace, Mr. Ron	1	Hallis, Dr. George
6	Dellmonache, Ms. Susan	1	Hallis, Dr. George
7	Dixon, Mr. Matt	1	Hallis, Dr. George
8	Edwards, Mr. Charles	1	Hallis, Dr. George
9	Edwards, Ms. Sonia	1	Hallis, Dr. George
10	Elsins, Ms. Marisa F.	1	Hallis, Dr. George
11	Griffin, Mr. Lantz	1	Hallis, Dr. George
12	Hall, Ms. Sharon	1	Hallis, Dr. George
13	Haubold, Ms. Ann	1	Hallis, Dr. George

Creating Macro Variables During PROC SQL Step Execution

The INTO Clause and the NOPRINT Option

You have seen how to create macro variables during DATA step execution. You can also create or update macro variables during the execution of a *PROC SQL step*. Remember that the SELECT statement in a PROC SQL step retrieves and displays data. The INTO clause in a SELECT statement enables you to create or update macro variables.

When you create or update macro variables during execution of a PROC SQL step, you might not want any output to be displayed. The PRINT | NOPPRINT option specifies whether a SELECT statement's results are displayed in output. PRINT is the default setting.

General form, PROC SQL with the NOPRINT option and the INTO clause:

```
PROC SQL NOPRINT;
    SELECT columnl<,column2,…>
           INTO :macro-variable-l<,:macro-variable-2,…>
           FROM table-1 \ view-1
           <WHEREexpression>
           <other clauses>;
```

QUIT;

where

columnl, column2,…

specifies one or more columns of the SQL table specified by *table-1* | *view-1*.

:macro-variable-l, :macro-variable-2,…

names the macro variables to create.

expression

produces a value that is used to subset the data.

other clauses

are other valid clauses that group, subset, or order the data.

Note Macro variable names are preceded by a colon.

Note For more information about PROC SQL, see the SAS documentation.

This form of the INTO clause does not trim leading or trailing blanks. Also, the INTO clause cannot be used when you create a table or a view.

Example

You can create a macro variable named **`totalfee`** that contains the total of all course fees, and use this macro variable in a later step. You use the NOPRINT option to suppress the output from the PROC SQL step.

```
proc sql noprint;
   select sum(fee) format=dollar10. into :totalfee
      from sasuser.all;
quit;
%let totalfee=&totalfee;

proc means data=sasuser.all sum maxdec=0;
   class course_title;
   var fee;
   title "Grand Total for All Courses Is &totalfee";
run;
```

Note This form of the INTO clause does not trim leading or trailing blanks, but the %LET statement removes any leading or trailing blanks that are stored in the value of **`totalfee.`**

The output from this PROC MEANS step shows the sum of all course fees in the DOLLARIO. format.

Grand Total for All Courses Is $354,380

The MEANS Procedure

Analysis Variable : Fee Course Fee

Description	N Obs	Sum
Artificial Intelligence	71	28400
Basic Telecommunications	69	54855
Computer Aided Design	66	105600

Database Design	77	28875
Local Area networks	74	48100
Structured Query Language	77	88550

Creating Variables with the INTO Clause

Earlier you learned how to create a series of related macro variables during execution of the DATA step by using the SYMPUT routine. Sometimes you might want to create a series of related macro variables during execution of a PROC SQL step. You can use the INTO clause to create one new macro variable for each row in the result of the SELECT statement.

General form, SELECT statement with the INTO clause for a range of macro variables:

```
PROC SQL NOPRINT;
      SELECT columnl
        INTO :macro-variable-l - :macro-variable-n
         FROM table-1 \ view-1
        <W~HEREexpression>
        <other clauses>;
```

QUIT;

where

columnl

specifies the column of the SQL table specified by table-1 | view-1.

:macro-variable-l - :macro-variable-n,…

names the macro variables to create.

expression

produces a value that is used to subset the data.

other clauses

are other valid clauses that group, subset, or order the data.

When storing values into a range of macro variables, or when using the SEPARATED BY option to store multiple values in one macro variable, the INTO clause of PROC SQL trims any leading and trailing blanks. Use the NOTRIM option If you want the blanks to be preserved. This treatment of leading and trailing blanks is in contrast to assigning the value of a DATA step variable for a macro variable in the SYMPUT routine on page 351.

Example

You can create a series of macro variables that contain the course code, location, and starting date of the first three courses that are scheduled in 2002. In this example, the macro variables **`crsid1-crsid3`** are assigned values of the data set variable **`Course_code`** from each of the first three rows of the PROC SQL result:

```
proc sql;
   select course_code, location, begin_date format=mmddyy10.
   into :crsid1-:crsid3,
        :place1-:place3,
        :date1-:date3
      from sasuser.schedule
      where year(begin_date)=2002
      order by begin_date;
quit;
```

This is the result of the PROC SQL step.

Course Code	Location	Begin

C003	Dallas	01/07/2002
C004	Boston	01/21/2002
C005	Seattle	02/25/200
C006	Dallas	03/25/2002

This is a representation of the symbol table after this PROC SQL step has run.

Global Symbol Table	
CRSIDl	*C003*
CRSID2	*C004*
CRSID3	*C005*
PLACE1	*Dallas*
PLACE2	*Boston*
PLACE3	*Seattie*
DATE1	*01/07/2002*
DATE2	*01/21/2002*
DATE3	*02/25/2002*

If you do not know how many macro variables will be created, you can issue a query to determine how many macro variables are needed and to create a macro variable to store that number. You can then run the query, using the macro variable as the suffix of the final macro variable in each series of macro variables.

Example

Suppose you want to create ranges of macro variables that contain the course code, location, and starting date of all courses that are scheduled in 2002. You do not know the number of courses. If you assign an arbitrarily large number as the suffix of the final macro variable range, only macro variables corresponding to the query result set are created. The macro variable SQLOBS is assigned a value reflecting the number of rows in the result set, matching the number of macro variables created in each range.

```
proc sql noprint;
   select course_code, location,
          begin_date format=mmddyy10.
      into :crsid1-:crsic 999,
           :place1-:place 999,
           :date1-:date999
      from sasuser.schedule
      where year(begin_date)=2002
      order by begin_date;
   %let numrows=&sqlobs;
   %put There are &numrows courses in 2002;
   %put _user_;
quit;
```

The SAS log shows that **`numrows`** is assigned a value of 4. The %PUT statement at the end of the program shows the names and values of all the macro variables that are created in the SELECT statement.

Table 10.7: SAS Log

```
114  proc sql noprint;
115    select course_code, location,
116        begin_date format=mmddyy10.
117      into :crsid1-:crsid999,
118        :place1-:place999,
119        :date1-:date999
120      from sasuser.schedule
121      where year(begin_date)=2002
122      order by begin_date;
123    %let numrows=&sqlobs;
124    %put There are &numrows courses in 2002;
```

```
There are 4 courses in 2002
125     %put _user_;
GLOBAL SQLOBS 4
GLOBAL CRSID2 C004
GLOBAL SQLOOPS 20
GLOBAL CRSID3 C005
GLOBAL DATE4 03/25/2002
GLOBAL PLACE1 Dallas
GLOBAL CRSID1 C003
GLOBAL PLACE2 Boston
GLOBAL PLACE3 Seattle
GLOBAL SYS_SQL_IP_ALL -1
GLOBAL SYS_SQL_IP_STMT
GLOBAL CRSNUM 3
GLOBAL DATE 01/08/2001
GLOBAL DATE1 01/07/2002
GLOBAL CRSID4 C006
GLOBAL DATE2 01/21/2002
GLOBAL DATE3 02/25/2002
GLOBAL NUMPAID 14
GLOBAL SQLXOBS 0
GLOBAL SQLRC 0
GLOBAL NUMROWS 4
GLOBAL NUMSTU 20
GLOBAL CRSNAME Local Area Networks
GLOBAL DUE $3,900
GLOBAL SQLEXITCODE 0
GLOBAL PLACE4 Dallas
126 quit;
```

Creating a Delimited List of Values

Sometimes, during execution of a PROC SQL step, you might want to create one macro variable that will hold all values of a certain data set variable. You can use an alternate form of the INTO clause in order to take all of the values of a column (variable) and *concatenate* them into the value of one macro variable.

General form, SELECT statement with INTO clause for combining values into one macro variable:

```
PROC SQL NOPRINT;
      SELECT columnl
           INTO :macro-variable-l
           SEPARATED BY 'delimiterl'
           FROM table-1 \ view-1
          <WHEREexpression>
          <other clauses>;
```

QUIT;

where

columnl

specifies the column of the SQL table specified by *table-1* | *view-1*.

:macro-variable-l

names the macro variable to create.

delimiterl

is enclosed in quotation marks and specifies the character that will be used as a delimiter in the value of the macro variable.

expression

produces a value that is used to subset the data.

other clauses

are other valid clauses that group, subset, or order the data.

This form of the INTO clause removes leading and trailing blanks from each value before performing the concatenation of values.

Example

You can use the SQL procedure to create one macro variable named **sites** that contains the names of all training centers that appear in the *Sasuser.Schedule* data set. The names will be separated by blanks.

```
proc sql noprint;
   select distinct location into :sites separated by ' '
      from sasuser.schedule;
quit;
```

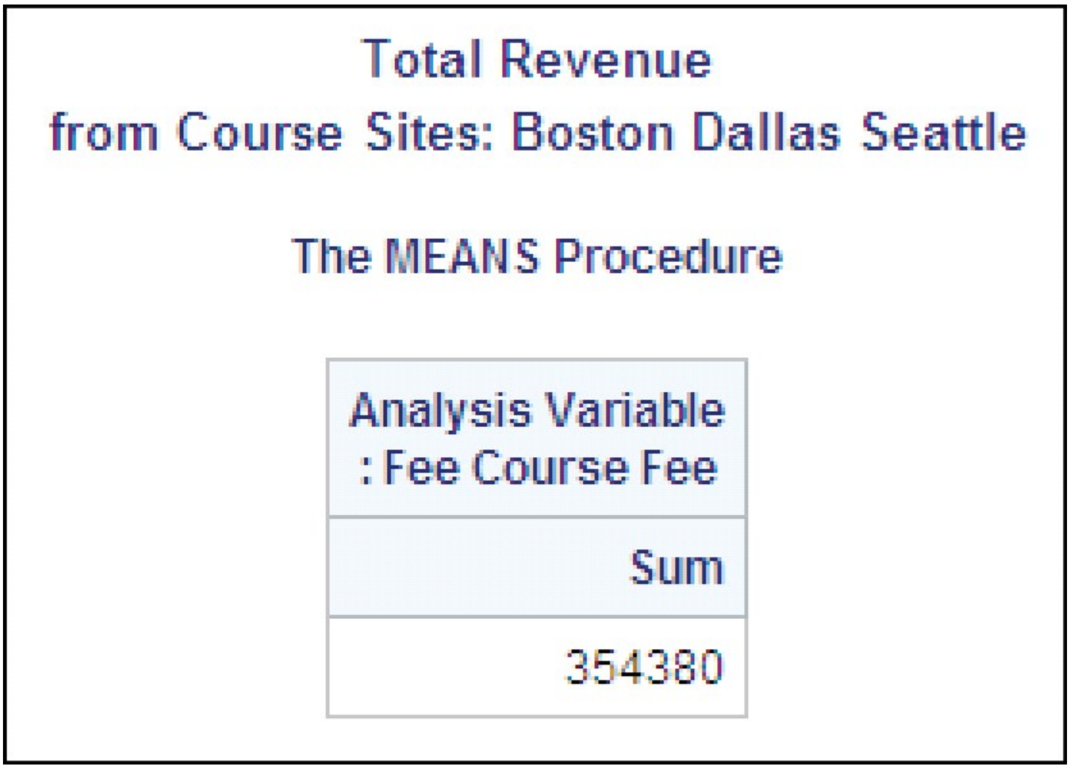
Here is a representation of the macro variable **sites** as it is stored in the global symbol table after this PROC SQL step has run.

Global symbol Table	
Sites	*Boston Dallas Seattle*

Now you can use the new macro variable in a title.

```
proc means data=sasuser.all sum maxdec=0;
   var fee;
   title1 'Total Revenue';
   title2 "from Course Sites: &sites";
run;
```

This is the output from the PROC MEANS step.



Total Revenue

from Course Sites: Boston Dallas Seattle

The MEANS Procedure

Analysis Variable : Fee Course Fee
Sum
354380

Working with PROC SQL Views

When you submit a PROC SQL step, the PROC SQL program code is placed into the input stack, and word scanning is performed for macro triggers in the same process as in other SAS programs.

In the following code, the macro variable reference **&crsid** is resolved during the creation of the PROC SQL view, resulting in a *constant value* whenever the view is used. For example, If the value of **crsid** is *C003* when this code is submitted, the view *Subcrsid* will always be based on the course code C003.

```
proc sql;
   create view subcrsid as
      select student_name, student_company,paid
         from sasuser.all
         where course_code="&crsid";
quit;
```

A better approach would be to use the SYMGET function to enable the view to look up the macro variable value. In the following example, the view *Subcrsid* is based on the value of **crsid** when the view is used:

```
proc sql;
   create view subcrsid as
      select student_name,student_company,paid
         from sasuser.all
         where course_code= symget('crsid');
quit;

%let crsid=C003;
proc print data=subcrsid noobs;
   title "Status of Students in Course Code &crsid";
run;

%let crsid=C004;
proc print data=subcrsid noobs;
   title "Status of Students in Course Code &crsid";
run;
```

PROC SQL does *not* perform automatic data conversion. You must use the INPUT function to convert the macro variable value to numeric if it is compared to a numeric variable.

The following code performs a query that is based on the numeric equivalent of the current value of the macro variable **crsnum.** The INPUT function is necessary in this WHERE statement because the value of the data set variable **Course_number** is numeric, but **crsnum** has a character value because it is a macro variable.

```
proc sql;
   create view subcnum as
      select student_name, student_company, paid
         from sasuser.all
         where course_number= input(symget('crsnum'),2.);
quit;

%let crsnum=4;
proc print data=subcnum noobs;
   title "Status of Students in Course Number &crsnum";
run;
```

Using Macro Variables in SCL Programs

Overview

SAS Component Language (SCL) programs are placed into the input stack, and word scanning is performed for macro triggers in the same process as in other SAS programs. Macro variable references that are outside of SUBMIT blocks are resolved before execution. Therefore, in the following example, a *constant value* will be compared to the SCL variable **Wage** during SCL execution:

```
MAIN:
   erroroff wage;
   if wage gt &max then erroron wage;
return;
```

Any text *within* a SUBMIT block is assumed to be SAS code and is therefore ignored by the SCL compiler when the SCL program is compiled. Macro variable references within SUBMIT blocks are not resolved until the SUBMIT block executes and the SAS code within the SUBMIT block is tokenized.

When a SUBMIT block executes, SAS attempts to resolve a macro variable reference **`(&name)`** to a corresponding SCL variable. If there is no corresponding SCL variable, the reference is passed to the macro processor for lookup in the global symbol table. You can force a reference **`(&name)`** within a SUBMIT block to be passed as a macro variable reference by preceding the name with two ampersands **`(&&name).`**

Also, there are several functions and routines that enable SCL programs and the macro facility to exchange information at execution time. We will examine these functions and routines.

You have already learned how to use the SYMPUT routine and the SYMGET function in a DATA step. Both the SYMPUT routine and the SYMGET function can be used in SCL programs. The syntax for each is exactly the same as it is in the DATA step.

Additionally, both the SYMPUT routine and the SYMGET function have numeric equivalents for use in SCL programs.

The SYMPUTN Routine

The SYMPUTN routine enables you to create a macro variable during execution of an SCL program and to assign a numeric value to it.

General form, SYMPUTN routine:

CALL SYMPUTN *('macro-variable', value);*

where

macro-variable

is the name of a global macro variable enclosed in single quotation marks with no ampersand. Alternatively, it is the name of an SCL variable *(not* enclosed in quotation marks) whose value is the name of a global macro variable.

value

is the numeric value that is assigned to *macro-variable*, which can be a number of the name of anumeric SCL variable.

Example

Suppose the SCL variable **`unitvar`** has a value of *unit* and the SCL variable **`unitnum`** has a numeric value of *200*. To create a macro variable whose name is the value of **`unitvar`** (in this case, *unit)* and assign a value equal to the value of the SCL variable **`unitnum`** (in this case, *200)* you submit the following statement within a SUBMIT block:

```
call symputn(unitvar, unitnum);
```

Similarly, to create a macro variable named **`unitvar`** and assign a numeric value of 500 to it, you submit the following statement within a SUBMIT block.

```
call symputn('unitvar', 500);
```

The SYMGETN Function

The SYMGETN function enables you to obtain the numeric value of a macro variable during execution of an SCL program.

General form, SYMGETN function:

SCL-variable **= SYMGETN** *('macro-variable');*

where

SCL-variable

is the name of a numeric SCL variable to which the value of *macro-variable* is assigned.

macro-variable

is the name of a global macro variable enclosed in single quotation marks with no ampersand. Alternatively, it is the name of an SCL variable *(not* enclosed in quotation marks) whose value is the name of a global macro variable.

Example

Suppose the SCL variable **`unitvar`** has avalué of *unit*, the macro variable **`unit`** has a value of *200*, and the macro variable **`unitvar`** has a value of *500*. The first statement below creates an SCL variable named **`unitnum`** and assigns to it a value of *200*. The second statement creates an SCL variable named **`unit`** and assigns it a value of *500*.

```
unitnum=symgetn(unitvar);
unit=symgetn('unitvar');
```

Note For more information about using macro variables in SCL, see the SAS documentation for the macro language.

Summary

Contents

This section contains the following topics.

Text Summary

Creating a Macro Variable during DATA Step Execution

When you create or update a macro variable with the %LET statement, all macro processing takes place before the execution of the DATA step. The SYMPUT routine enables you to create or update macro variables during DATA step execution. Depending on how the arguments are coded, you can create either a single macro variable or multiple macro variables. You can use the SYMPUT routine with literal strings to create a macro variable and to assign either an exact name or an exact text value to it. You can use the SYMPUT routine with a DATA step variable to assign the value of that DATA step variable to a macro variable.

You can use the SYMPUTX routine to create or update a macro variable during DATA step execution, and to automatically strip leading and trailing blanks from the macro variable name and value. You can also use a DATA step expression as an argument to the SYMPUT routine in order to apply DATA step functions to a value before you assign that value to a macro variable. The PUT function is often useful in conjunction with the SYMPUT and SYMPUTX routines.

Creating Multiple Macro Variables during DATA Step Execution

You can use the SYMPUT or SYMPUTX routine with two DATA step expressions to create a series of related macro variables within one DATA step.

Referencing Macro Variables Indirectly

Sometimes, it is useful to use indirect references to macro variables. For example, you might want to use a macro variable to construct the name of another macro variable. You can reference a macro variable indirectly by preceding the macro variable name with two or more ampersands.

Obtaining Macro Variable Values during DATA Step Execution

The SYMGET function is used by both the DATA step and the SQL procedure to obtain the value of a macro variable during execution. You can use the SYMGET function to assign a macro variable value to a DATA step variable.

Creating Macro Variables during PROC SQL Step Execution

You can access the macro facility in a PROC SQL step by using the INTO clause in the SELECT statement. Various forms of the INTO clause enable you to create a series of macro variables, a varying number of macro variables, or a single macro variable that records a value that is created by concatenating the unique values of an SQL variable. You can use the NOPRINT option to prevent a PROC SQL step from creating output.

Working with PROC SQL Views

When you submit a PROC SQL step, the PROC SQL program code is placed into the input stack, and word scanning is performed for macro triggers in the same process as in other SAS programs.

Using Macro Variables in SCL Programs

SAS Component Language (SCL) also has routines and functions that assign values to macro variables and that obtain values from a macro symbol table. The SYMPUT routine and the SYMGET function can be used in an SCL program in the same way that they can be used in a DATA step program. Also, the SYMPUTN routine can be used to create macro variables and to assign numeric values to those variables during the execution of an SCL program. The SYMGETN function can be used to obtain the numeric value of a macro variable during the execution of an SCL program.

Syntax

```
CALL SYMPUT(macro-variable, text);
PUT(source, format.)
CALL SYMPUT(expressionl, expression2);
CALL SYMPUTN ('macro-variable', value)·,
CALL SYMPUTX(macro-variable, text);
SYMGET (macro-variable)
SYMGETN('macro-variable')
PROC SQL NOPRINT;
     SELECT columnl,<column2,…>
          INTO :macro-variable-l<,\macro-variable-2,…>
          FROM table-1 | view-1
          <WHEREexpresson>
          <other clauses>;
QUIT;
PROC SQL NOPRINT;
SELECT column1,…>
          INTO :macro-variable-l - :macro-variable-n
          FROM table-1 | view-1
          <WHEREexpresson>
          <other clauses>;
QUIT;
PROC SQL NOPRINT;
     SELECT columnl
          INTO :macro-variable-l
          SEPARATED BY 'delimiterl'
          FROM table-1 | view-1 <WREREexpression>
         <other clauses>\
QUIT;
```

Sample Programs

Using CALL SYMPUT to Create Macro Variables

```
options symbolgen pagesize=3 0;
%let crsnum=3;
data revenue;
   set sasuser.all end=final;
   where course_number=&crsnum;
   total+1;
   if paid='Y' then paidup+1;
   if final then do;
      if paidup<total then do;
         call symput('foot','Some Fees Are Unpaid');
      end;
      else do;
         call symput('foot','All Students Have Paid');
```

```
     end;
  end;
run;
proc print data=revenue;
   var student_name student_company paid;
   title "Payment Status for Course &crsnum";
   footnote "&foot";
run;
```

Referencing Macro Variables Indirectly

```
options symbolgen;
data _null_;
    set sasuser.courses;
    call symput(course_code, trim(course_title));
run;

%let crsid=C005;
proc print data=sasuser.schedule noobs label;
    where course_code="&crsid";
    var location begin_date teacher;
    title1 "Schedule for &&&crsid";
run;

%let crsid=C002;
proc print data=sasuser.schedule noobs label;
   where course_code="&crsid";
   var location begin_date teacher;
   title1 "Schedule for &&&crsid";
run;
```

Using SYMGET to Obtain Macro Variable Values

```
data teachers;
   set sasuser.register;
   length Teacher $ 20;
   teacher=symget('teach'||left(course_number));
run;

proc print data=teachers;
   var student_name course_number teacher;
title1 "Teacher for Each Registered Student";
run;
```

Creating Macro Variables with the INTO Clause

```
proc sql noprint;
   select course_code, location, begin_date format=mmddyy10.
      into :crsid1-:crsid3,
           :place1-:place3,
           :date1-:date3
      from sasuser.schedule
      where year(begin_date)=2002
      order by begin_date;
quit;
```

Points to Remember

- The SYMPUT routine can be used to create or update macro variables during DATA step execution.
- The values of macro variables are always character values. In the DATA step, SYMPUT performs automatic numeric to character conversion on any numeric value that you attempt to assign to a macro variable.
- The SYMGET function can be used to obtain the value of a macro variable during the execution of a DATA step, a PROC SQL step, or an SCL program.
- The INTO clause can be used in the SELECT statement to create or update macro variables during execution of aPROC SQL step.
- The SYMPUT and SYMPUTN routines can be used to create or update macro variables during the execution of an SCL program.

Quiz

Select the best answer for each question. After completing the quiz, check your answers using the answer key in the appendix.

1. Which of the following is false? ?

a. A %LET statement causes the macro processor to create a macro variable before the program is compiled.

b. To create a macro variable that is based on data calculated by the DATA step, you use the SYMPUT function.

c. Macro functions are always processed during the execution of the DATA step.

d. Macro variable references in a DATA step are always resolved before DATA step execution.

2. Which of the following correctly creates a macro variable named **`region`** and assigns to it a value that is based on the value of the data set variable **`Location?`** ?

a.
```
a. data new;
     set sasuser.all;
     if location='Boston' then do;
        call symput('region', 'East');
     end;
     else do;
        call symput('region', 'West');
     end;
   run;
```

b.
```
b. data new;
     set sasuser.all;
     if location='Boston' then do;
        %let region=East;
      end;
      else
         %let region=West;
      end;
   run;
```

c.
```
c. data new;
     set sasuser.all;
     if location='Boston' then do;
        call symput(region, "East");
     end;
     else
        call symput(region, "West");
     end;
   run;
```

d.
```
d. data new;
     set sasuser.all;
     if location='Boston' then do;
        symput(region, East);
     end;
     else
        symput(region, West);
     end;
   run;
```

3. The SYMPUT routine cannot ?

a. be used to assign a data set variable as a value to a macro variable.

b. create a series of macro variables in one DATA step.

c. automatically convert a numeric value to a character value when used to assign a value to a macro variable in a DATA step.

d. be used to assign a numeric value to a macro variable in an SCL program.

4. Which of the following programs correctly creates a series of macro variables whose names are values of the data set variable **Course_code**, then indirectly references one of those macro variables in a later step? ?

a.
```
a. data _null_;
      set sasuser.courses;
      call symput(course_code, trim(course_title));
   %let crsid=C005;
   proc print data=sasuser.schedule noobs label;
     where course_code="&crsid";
     var location begin_date teacher;
     title1 "Schedule for &c005";
   run;
```

b.
```
b. data _null_;
      set sasuser.courses;
      call symput(course_code, trim(course_title));
   run;
   %let crsid=C005;
   proc print data=sasuser.schedule noobs label;
     where course_code="&crsid";
     var location begin_date teacher;
     title1 "Schedule for &&&crsid";
   run;
```

c.
```
c. data _null_;
     set sasuser.courses;
     call symput('course_code', trim(course_title));
   run;
   %let crsid=C005;
   proc print data=sasuser.schedule noobs label;
     where course_code="&crsid";
     var location begin_date teacher;
     title1 "Schedule for &&&crsid";
   run;
```

d.
```
d. data _null_;
     set sasuser.courses;
     call symget(course_code, trim(course_title));
    run;

    %let crsid=C0 05;
   proc print data=sasuser.schedule noobs label;
     where course_code="&crsid";
     var location begin_date teacher;
     title1 "Schedule for &&&crsid";
   run;
```

5. Which of the following statements about the resolution of macro variable references is false? ?

a. Two ampersands resolve to one ampersand.

b. If more than four consecutive ampersands precede a name token, the macro processor generates an error message.

c. Re-scanning continues until there are no remaining macro triggers that the macro processor can resolve.

d. The macro processor always re-scans a name token that is preceded by multiple ampersands or by multiple percent signs.

6. In which of the following situations would you use SYMGET rather than a macro variable reference **`(&macvar)?`** ?

a. to create a DATA step variable from a macro variable value during the execution of the DATA step

b. to include a macro variable reference in a PROC SQL view

c. to access the value of a macro variable during the execution of an SCL program

d. all of the above

7. Which of the following correctly creates a macro variable in a PROC SQL step? ?

a. `call symput(daily_fee, put(fee/days, dollar8.);`

b. `%let daily_fee=put(fee/days, dollar8.)`

c. `select fee/days format=dollar8. into :daily_fee from sasuser.all;`

d. `select fee/days format=dollar8. into daily_fee from sasuser.all;`

8. According to the global symbol table shown here, what value will a reference to **`&&teach&crs`** resolve to? ?

Global Symbol Table	
TEACH1	*Hallis, Dr.George*
TEACH2	*Wickam, Dr.Alice*
TEACH3	*Forest, Mr.Peter*
CRS	*3*

a. &TEACH3

b. TEACH3

c. Forest, Mr. Peter

d. none of the above

9. Which of the following statements correctly creates a DATA step variable named **`Price`** and assigns to it the value of the macro variable **`daily_fee`** during DATA step execution? ?

a. `price=&daily_fee;`

b. `price=symget(daily_fee);`

c. `price=symget(&daily_fee);`

d. `price=symget("daily_fee");`

10. Which of the following is false? ?

a. The SYMPUT routine can be used to create a macro variable during execution of the DATA step or during execution of an SCL program.

b. In the DATA step, the SYMPUT routine automatically converts to a character value any numeric value that you attempt to assign as the value of a macro variable.

c. PROC SQL automatically converts to a numeric value any macro variable value that you attempt to compare to a numeric value.

d. In an SCL program, the SYMPUTN routine can be used to assign a numeric value to a macro variable.

Answers

1. Correct answer: c

   Most macro functions are handled by the macro processor before any SAS language statements in the DATA step are executed. For example, the %LET statement and any macro variable references i&macvar) are passed to the macro processor before the program is compiled. In order to create or update macro variables during DATA step execution, you use the SYMPUT routine.

2. Correct answer: a

   To create a macro variable and assign to it a value that is based on the value of a DATA step variable, you use the SYMPUT routine. In the SYMPUT routine, to assign a literal string as a macro variable name, you enclose the literal in quotation marks. To assign a literal string as a value of the macro variable, you enclose the literal in quotation marks.

3. Correct answer: d

   The SYMPUT routine enables you to assign a data set variable as the value of a macro variable. You can also use the SYMPUT routine to create a series of related macro variables. Because all macro variable values are character strings, SYMPUT automatically converts any numeric value that you attempt to assign as a value for a macro variable. In an SCL program, you must use SYMPUTN rather than SYMPUT if you are attempting to assign a numeric value to a macro variable.

4. Correct answer: b

   You can use multiple ampersands to create an indirect reference when the value of one macro variable is the name of another. If you enclose the DATA step variable name in quotation marks in the SYMPUT routine, the new macro variable will have the same name as the DATA step variable rather than having the DATA step variable's value as a name. Use the SYMGET function to obtain the value of a macro variable during the execution of a DATA step.

5. Correct answer: b

   If more than four consecutive ampersands precede a name token, rescanning continues from left to right until no more triggers can be resolved. The Forward Rescan ride describes how the macro processor resolves macro variable references that start with multiple ampersands or with multiple percent signs.

6. Correct answer: d

   A macro variable reference `(&macvar)` is resolved before any SAS language statements are sent to the compiler. The SYMGET function enables you to obtain the value of a macro variable during the execution of a DATA step or a PROC SQL step. The SYMGET function can also be used to obtain the value of a macro variable during the execution of an SCL program.

7. Correct answer: c

   To create a macro variable during the execution of a PROC SQL step, use the INTO clause of the SELECT statement. In the INTO clause, you precede the name of the macro variable with a colon.

8. Correct answer: c

   You can use multiple ampersands to delay the resolution of a macro variable reference. You can also combine macro variable references in order to create new tokens. In this example, the reference `&&teach&crs` resolves to `&teach3` on the first scan. On the next scan, `&teach3` resolves to *Forest, Mr. Peter*.

9. Correct answer: d

   You can use the SYMGET function in an assignment statement to obtain the current value of a macro variable and to assign that value to a DATA step variable. The SYMGET function enables you to obtain the value of a macro variable during execution of a DATA step, a PROC SQL step, or an SCL program.

10. Correct answer: c

    The SYMPUT routine can be used in either the DATA step or in an SCL program. In the DATA step, the SYMPUT routine will perform automatic conversion on numeric values that you attempt to assign as values for macro variables, using the BEST12. format. In an SCL program, you should use the SYMPUTN routine if you want to assign a numeric value as a value for a macro variable. In a PROC SQL step, you need to use the INPUT function in order to convert macro variable values to numeric before you compare them to other numeric values.